NIKTECH INC

IP Cores Users' Guide

NIKTECH INC

IP Cores Users Guide

190 Shooting Star Isle
Foster City, CA - 94404

Table of Contents

Introduction

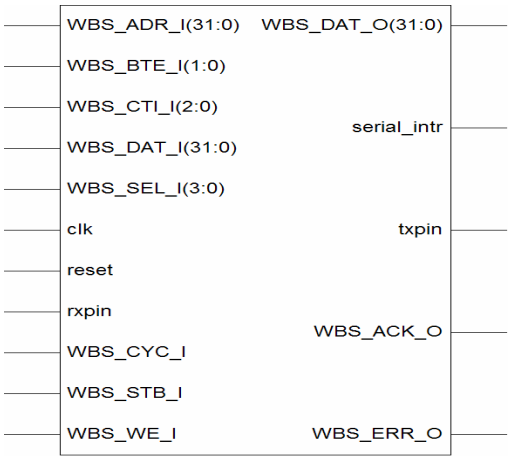
This document describes the IP cores that are provided with MANIK 32bit RISC core and the software usage model (if applicable).

UART (Serial Port).

The UART IP core provides a WISHBONE compliant interface to a standard RS232 connection. The following is a block diagram of the core. The source code for the core can be found in *$(MANIK_BASE)/vhdl/cores/serial.vhd*.

The Wishbone Signals are prefixed with ***WBS_.*** The ***txpin*** is the output and must be connected to the RS232C driver, the ***rxpin*** in the input pin and should also be connected to the RS232 driver.

Block Diagram



Configuration Parameters

Name	Type	Default Value	**Description**
WIDTH	**Integer**	**32**	Width of Wishbone Bus (should not be changed)
BAUD_RATE	**Integer**	**115200**	The default/startup baud rate.
CORE_FREQ_MHZ	**Integer**	**50**	The frequency of the input wishbone clock in MHZ. The baudrate is computed based on this clock.

Register Map

Name	Offset	Read/Write	Bits	**Description**
Receive/Transmit	**0**	**R/W**	**8**	A write operation to this register will queue a character into the transmit buffer. If the transmit buffer is not empty the write operation will block till it becomes empty. A read operation from this register will return the character received. If the receive buffer is empty the read operation will block till a character is received.
Control/Status	**1**	**R/W**	**8**	Read operation returns the status. Bit-0 – Transmit buffer empty (R/O) Bit-1 – Transmit buffer full (R/O) Bit-2 – Receive buffer full (R/O) Bit-3 – Receive data available (R/O) Bit-4 – Polled mode; Interrupt disabled (R/W) Bit-5 – Overrun – error (R/O) Bit-6 – Framing error (R/O)
Clock Divisor	**4**	**W/O**	**32**	Clock divisor.

Theory of operation

The serial port consists of a transmit module and a receive module. The baud rate is generated by dividing the WISHBONE clock to generate a BAUD RATE * 16 clock. The baud rate can be provided as a configuration option or can be programmed by software. The following formula is used to compute the divisor value.

```
Divisor = ((Baud<<27)+(ClkFrequency>>5))/(ClkFrequency>>4);
```

The Data and Stop bits are fixed to 8 and 1 respectively and cannot be changed. The read and write are blocking. The interrupt is generated when the polled bit is zero and a character is ready in the receive buffer; the transmit module cannot generate an interrupt.

Software support.

The following functions are provided to interface with the UART IP core. The routines assume that a macro UART_BASE has been defined, this is the base address for the UART core.

`char ser_get_stat()` – The function returns the status register of the UART. The description of the register bits are provided above.

`void ser_set_polled(int)` – The function is used to set or unset the polled bit in the control register. A non-zero value as argument will set the polled bit, a zero value will clear the polled bit.

`void ser_set_baud(int)` – The function will write the value provided as argument to the ***Divisor*** register. The value of the divisor should be calculated using the formula given in the previous section.

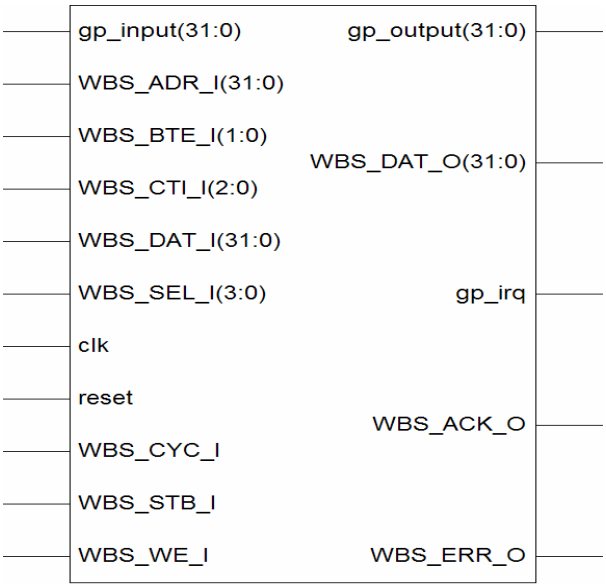
`void ser_put(char)` – Writes the value provided as argument to the transmit buffer . The function call will block till the transmit buffer is empty. If a non-blocking access is required then the software should read the status register to determine if the transmit buffer is empty before it writes to it.

`char ser_get()` – Returns the received character from the UART's receive buffer. The routine will block if there are no characters available.

GPIO (General Purpose I/O)

The GPIO module, provides a WISHBONE interface to a configurable number for I/O ports. The GPIO can be used to simultaneously output and input (via separate ports). The input ports can be configured to detect edges or levels. The module is also capable of generating interrupts on predefined levels or edges on the input port. The source code for the core can be found in *$(MANIK_BASE)/vhdl/cores/gpio.vhd*.

Block Diagram



Configuration Parameters

Name	Type	Default Value	**Description**
WIDTH	**Integer**	**32**	Width of Wishbone Bus (should not be changed)
I_WIDTH	**Integer**	**32**	Width of input port
O_WIDTH	**Integer**	**32**	Width of output port
DEBOUNCE	**Boolean**	**False**	Currently not implemented
GENIRQ	**Boolean**	**False**	Generate interrupt on input (depending on input type)
I_TYPE	**Integer**	**1**	Input Type

			1 – Level ;reading input port will return the level on the input pin. If GENIRQ is true, interrupt will be generated if any of the inputs is '1'. 2- Positive edge; Reading input port will retrun 1 for those input port that have transitioned from zero to 1. If GENIRQ is true then an interrupt will be generated when any of the input ports transition from 0 to 1. 3-- Negative edge; Reading input port will retrun 1 for those input port that have transitioned from 1 to 0. If GENIRQ is true then an interrupt will be generated when any of the input ports transition from 1 to 0. 4 – Either edge; Reading input port will retrun 1 for those input port that have transitioned from 1 to 0 or from 0 to 1. If GENIRQ is true then an interrupt will be generated when any of the input ports transition from 1 to 0 or from 0 to 1.

Register Map

Name	Offset	Read/Write	Bits	**Description**
Input/Output	**0**	**R/W**	**WIDTH**	A write operation to this register update the output port. A read operation from this register will return the value from the input port.
IRQ Mask	**4**	**R/W**	**WIDTH**	The value in this register determines the bits that will used to detect an interrupt. Only the bits that are set to '1' will be used for the interrupt detection. Writing a zero to a bit will also clear the corresponding bit in input capture register
IRQ Detect	**8**	**R/O**	**WIDTH**	Will return the input edge capture register. The bits which had the requested transition will be set to 1 in this register.

Theory of operation

The GPIO core consists of an output register (to drive the output pins) and an input register (to capture the values of the input pins). The input pins are double buffered. The GPIO core can be configured to detect the Level (I_TYPE = 1), rising edge (I_TYPE=2), falling edge (I_TYPE=3) or any edge (I_TYPE = 4). It can also be configured to generate an interrupt when one of these events occur..

Output

The output IO pins of the GPIO core (gp_output) is controlled by writing the *Input/Output Register* at offset 0. The value once written cannot be read back, the software should maintain a copy of it if older values are required. The power-up & reset values for the output registers is zero.

Input

The value of the input pins of the GPIO core can be sampled by reading the *Input/Output register* at offset 0. The inputs from the **pads** are double buffered. The core can be configured to generate an interrupt (GENIRQ configuration parameter). The software can control the interrupt generation by writing to the IRQ Mask register at offset 4. Clearing a bit in the IRQ mask register also clears the corresponding bit in the edge detection register (allowing it to capture the next edge). The recommended steps for generating and handling interrupts from the GPIO core are

a) Generate core with GENIRQ = true, and the desired I_TYPE parameter.

b) Set the desired bits in IRQ Mask to '1'.

c) An interrupt will be generated when

   i. I_TYPE = Level (1). When the level of an input put pin 1 and the corresponding bit in the IRQ Mask is 1.

   ii. I_TYPE = Rising Edge (2). When the level of an input put pin transitions from 0 to 1 and the corresponding bit in the IRQ Mask is 1.

   iii. I_TYPE = Falling Edge (3). When the level of an input put pin transitions from 1 to 0 and the corresponding bit in the IRQ Mask is 1.

   iv. I_TYPE = Either Edge (4). When the level of an input put pin transitions from 0 to 1 or from 0 to 1 and the corresponding bit in the IRQ Mask is 1.

d) The interrupt service routine should read the IRQ Detect register (offset 8) and determine the bit that caused the interrupt. The IRQ Mask register should be updated with this bit set to 0 this will clear the interrupt. The software can then re-enable the interrupt by updating the IRQ Mask register bit to 1.

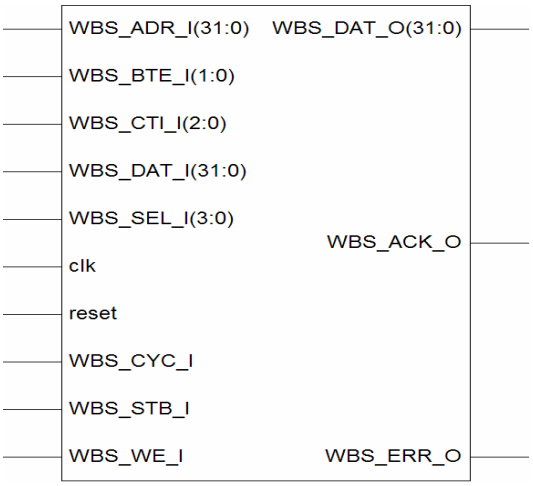
Software support.

The GPIO core has no software support. See MANIK-Software Developers Guide for details on registering and handling interrupts.

On-Chip RAM

This CORE provides a WISHBONE compliant interface to On-chip memory. The memory can be initialized with some values, the initialization process is vendor specific. The source code the core can be found in *$(MANIK_BASE)/vhdl/cores/ocsyncram.vhd*.

Block Diagram



Configuration Parameters

Name	Type	Default Value	**Description**
WIDTH	**Integer**	**32**	Width of Wishbone Bus (should not be changed)
ADDR_WIDTH	**Integer**	**32**	Width of Wishbone Address Bus (Should not be changed)
RAM_AWIDTH	**Integer**	**11**	Width of RAM address bus
RAM_INITFILE	**String**	**“”**	The Name of the initialization file. The format of the initialization file is vendor and synthesis tool specific.

Register Map

Not applicable.

Theory Of Operation

The core instantiated vendor specific memory elements to provide the desired size of memory. The memory is byte addressable, the core will perform a read modify write if the vendor specific memory element does not provide byte access.

Xilinx – Initialization

The Xilinx synthesis tool (XST) provides **VHDL Textio** interface to read memory initialization values from a file. The initialization routine requires the file to be in a format that be read by the Textio routines. The steps to create the initialization files are

a) Create the application ELF file (.elf).

b) Use objcopy to convert the ELF file to a binary file.

```
manik-elf-objcopy -O binary <Application>.elf
<Application>.bin
```

c) Use *conv2bin* utility to convert the binary file to Textio readable format.

```
conv2bin <Application>.bin <Application>.mem
```

The .mem file created can be used as the RAM_INITFILE parameter for the memory.

Note: Synplicity (Synplify) does NOT support this method of memory initialization.

Altera – Initialization

Memories for Altera are created by instantiated using the Altera provided **altsyncram** (lpm). This lpm can take an Intel Hex file formatted file to initialize the memory. The Intel hex file needs to be in a specific format for the initialization to work correctly. The steps to create the initialization file are

a) Create the application ELF file (.elf).

b) Use objcopy to convert the ELF file to Intel Hex format. `manik-elf-objcopy -O hex <Application>.elf <Application>.hex`

c) Use Altera Quartus-II to convert this Intel Hex file to a format that the Altera tools will understand.

   i. Start Quartus-II and open the Intel Hex file. **File -> Open (Select file type .mem)**

   ii. A dialog box will appear enter **Word Size: *8***

iii. Then open **. Edit -> Memory Size Wizard …**

iv. Change **Word Size : 32** then click **Next >.**

v. Select radio button **Combine existing Words.** Then click **Next >** then **Finish**

vi. Save the file

This will modify the Intel hex format to be compatible with the initialization required for the memory. Note if you recompile the application the process has to be repeated.

Vendor neutral Initialization

The MANIK system comes with an utility that will create a .vhdl file with generic memory initialization that vendor independent synthesis tools such as **Synplify** as well as vendor specific tools can infer initialized RAMs. The steps to create such file are

a) Create the application ELF file (.elf).

b) Use objcopy to convert the ELF file to a binary file.

manik-elf-objcopy –O binary <Application>.elf <Application>.bin

c) Use utility **gen_vhdl_ram** to create an initialized memory file

gen_vhdl_ram <Application>.bin <Application>.vhd <Application>

The first parameter is the input file, the second is the output file and third is the name of the vhdl **entity** name.

The source code for the utility is provided with the package.

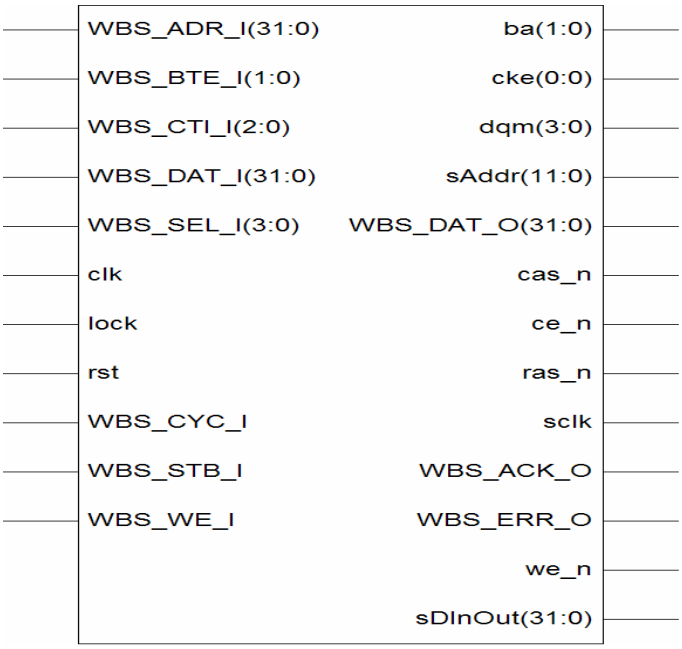
Software Support

Not Applicable.

SDRAM Controller

The basic SDRAM controller is available from XESS Corporation, a WISHBONE interface was added to the memory controller, some of the vendor specific code was converted to generic vhdl.

Block Diagram



Configuration Parameters

Name	Type	Default Value	**Description**
WIDTH	**Integer**	**32**	Width of Wishbone Bus (should not be changed)
ADDR_WIDTH	**Integer**	**32**	Width of Wishbone Address Bus (Should not be changed)
FREQ	**Integer**	**50000**	Frequency of wishbone clock in KHz
PIPE_EN	**Boolean**	**False**	Enables pipelining for read (True NOT Tested)
MAX_NOP	**Natural**	**1000**	Number of NOPS before starting self-refresh
MULTI_ACT_ROWS	**Boolean**	**False**	True will allow an active row in each bank
CAS_LATENCY	**Integer**	**3**	CAS latency of SDRAM

NROWS	**Integer**	**4096**	Number of ROWS in the SDRAM Array
NCOLS	**Integer**	**256**	Number of Columns in the SDRAM Array
RAM_ADDR_WIDTH	**Integer**	**12**	Number of address bits for the SDRAM
SDRAM_CKES	**Integer**	**1**	Number for CKEs signals (For multiple banks)

Register Map

Not applicable.

Theory of operation

Refer to the document http://www.xess.com/appnotes/an-071205-xsasdramcntl.pdf for details of the operation.

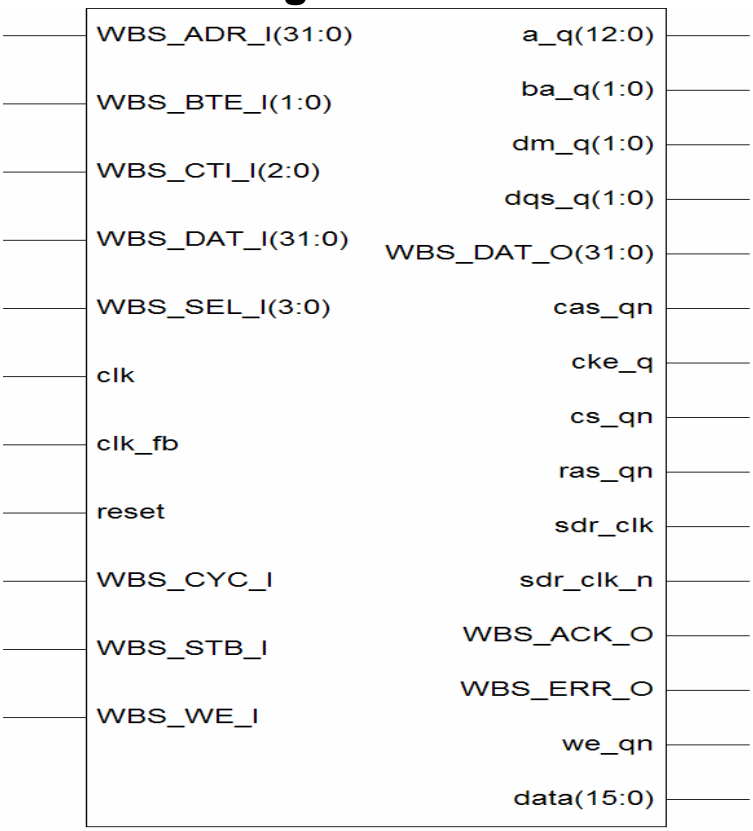
Software Support

Not Applicable.

DDR SDRAM Controller

The basic DDR SDRAM controller is the available from http://www.opencores.org . A WISHBONE interface was added to the controller. Support for ***dm*** signal has also been added.

Block Diagram



Configuration parameters

Name	Type	Default Value	**Description**
WIDTH	**Integer**	**32**	Width of Wishbone Bus (should not be changed)
ADDR_WIDTH	**Integer**	**32**	Width of Wishbone Address Bus (Should not be changed)
FREQ_KHZ	**Positive**	**50_000**	Frequency of Wishbone clock in KHz
DDR_DM_WIDTH	**Positive**	**2**	Width of DM Signal
DDR_DQS_WIDTH	**Positive**	**2**	Width of DQS Signal
DDR_DATA_WIDTH	**Positive**	**16**	DDR Module DATA Width
DDR_ADDR_WIDTH	**Positive**	**13**	DDR Module Address width
DDR_BANK_WIDTH	**Positive**	**2**	Number of Bank address lines
AUTO_PRECHARGE	**Positive**	**10**	Bit position in column address for auto

			precharge

Register Map

Not Applicable

Theory of operations

The core uses many XILINX specific components and is tested on a XILINX platform only. It uses two DCM's . For more information refer to documentation for the core on http://www.opencores.org.

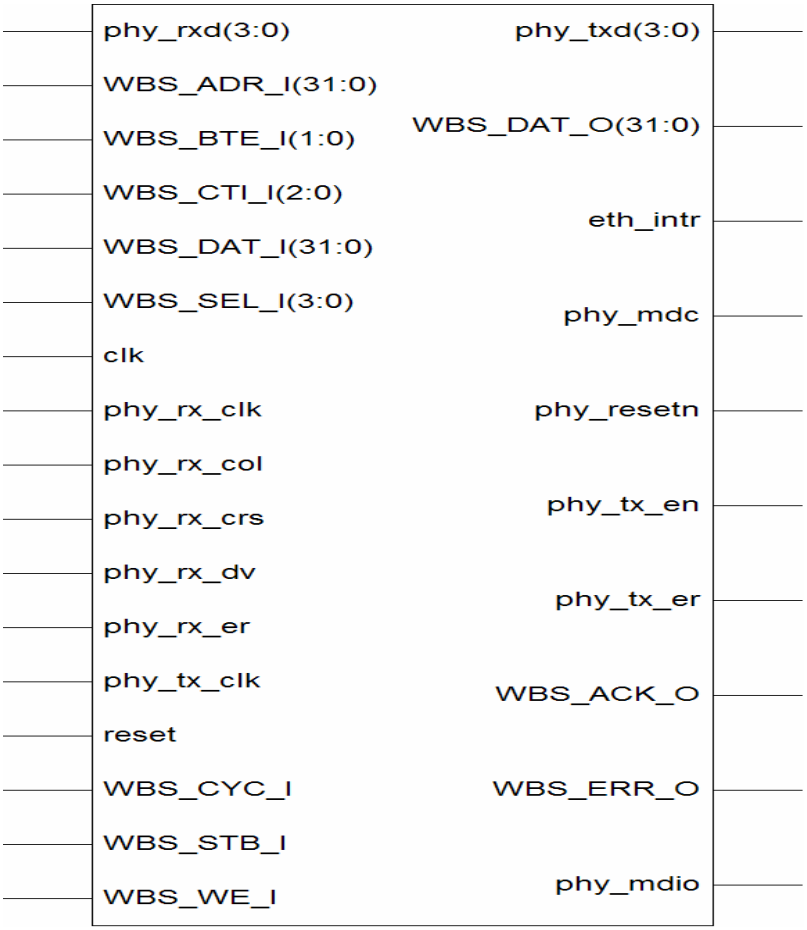
Software Support

Not Applicable

EasyMAC (10BaseT Ethernet MAC)

The core is a small Wishbone compliant Ethernet MAC. It is capable of operating at 10BaseT. The core is designed to have a small foot print. The source code for core can be found in *$(MANIK_BASE)/vhdl/cores/eth_mac.vhd*.

Block Diagram



Configuration Parameter

Name	Type	Default Value	**Description**
WIDTH	**Integer**	**32**	Width of Wishbone Bus (should not be changed)
ADDR_WIDTH	**Integer**	**32**	Width of Wishbone Address Bus (Should not be changed)
ETH_ADDR_AWIDTH	**Integer**	**11**	Address width for internal FIFO. (Should not be changed)
DEFAULT_MAC_ADDRESS	**Std_logic_vector**	**0**	The default Mac address .

Register Map

<table>
<tr><th>Name</th><th>Offset</th><th>Read/ Write</th><th>Bits</th><th colspan="3">Description</th></tr>
<tr><td>Control/Status Register</td><td>0</td><td>R/W</td><td>WIDTH</td><td>Bit(s)</td><td>R/W</td><td>Description</td></tr>
<tr><td></td><td></td><td></td><td></td><td>0</td><td>W/O</td><td>Resets the write FIFO pointer for the transmit buffer</td></tr>
<tr><td></td><td></td><td></td><td></td><td>1</td><td>R/W</td><td>Receive enable</td></tr>
<tr><td></td><td></td><td></td><td></td><td>2</td><td>R/W</td><td>Read - Transmit busy

Write – Start Transmitting</td></tr>
<tr><td></td><td></td><td></td><td></td><td>3</td><td>W/O</td><td>Reset MAC address pointer</td></tr>
<tr><td></td><td></td><td></td><td></td><td>4</td><td>W/O</td><td>Read from Receive FIFO complete</td></tr>
<tr><td></td><td></td><td></td><td></td><td>5</td><td>R/W</td><td>Receive interrupt enabled(1)/disabled(0)</td></tr>
<tr><td></td><td></td><td></td><td></td><td>6</td><td>R/W</td><td>Transmit interrupt enabled(1)/disabled(0)</td></tr>
<tr><td></td><td></td><td></td><td></td><td>7</td><td>R/W</td><td>Enable(1)/Disable(0) Promiscuous mode</td></tr>
<tr><td></td><td></td><td></td><td></td><td>8</td><td>R/O</td><td>Receive packet waiting in FIFO</td></tr>
<tr><td></td><td></td><td></td><td></td><td>9</td><td>R/O</td><td>Received packet has CRC error (valid when bit 8 is 1)</td></tr>
<tr><td></td><td></td><td></td><td></td><td>10:15</td><td></td><td>Reserved</td></tr>
<tr><td></td><td></td><td></td><td></td><td>31:16</td><td>R/O</td><td>Length of received packet (valid when bit 8 is 1)</td></tr>
<tr><td>Data Register</td><td>4</td><td>R/W</td><td>WIDTH</td><td colspan="3">This register is used to Read from the receive FIFO or to write to the transmit FIFO. Note only lower 8 bits contain valid data.</td></tr>
<tr><td>MAC Address Register</td><td>8</td><td>R/W</td><td>WIDTH</td><td colspan="3">This register is used to read/update the MAC Address. Only the lower 8 bits contain valid data, six read/write operations need to be performed to get/update the entire</td></tr>
</table>

				MAC address.
Counter Register	**12**	**R/O**	**WIDTH**	The counter returns the number of invalid frames received from the PHY interface.

Theory of operation

The EasyMAC core is designed to have small hardware footprint and to have a easy software interface. The core consists of three modules, the MAC address module, the receive module & transmit module.

MAC Address Module

This module maintains the MAC address of the core. The default value for the MAC address is all zeros. The read or write the MAC address the follow the step.

a) Set the Reset MAC Address pointer bit (3) in the Control register (Offset 0).

b) Read/Write the MAC address Register (offset 8). The read or write does a 32 bit load/store however only the lowest order 8 bits are read/written. The MAC address is read/written lowest order byte first. The core assumes that the MAC address is 6 bytes (48 bits) and auto increments the MAC address pointer after each read/write. Reading/Writing more than 6 bytes will result in undefined behaviour.

Receive Module

The receive module will receive packets when the **receive enable** bit (1) is set to 1. It will receive packets with MAC destination address that matches the programmed MAC address or a **broadcast** packet; when the promiscuous mode is set all packets received will be received.

The receive module writes the packets to a FIFO (2Kbytes), when a complete packet has been received the module will set the Packet Received (bit 8) in the Control register; if a CRC error was detected the module will also set the CRC error flag. An interrupt will be generated if the Receive Interrupt bit (5) is set in the control register.

Note no other packets will be received till the processor empties the FIFO and writes a 1 into the Read from FIFO complete (bit 4) in the control register. The software should use the following steps to receive a packet from the receive FIFO.

a) Wait for packet to arrive (either receive interrupt or polling the receive flag in the control register).

b) Get the length of the received packet. (Higher order 16 bits of the control register)

c) Read the data from the FIFO (one byte at a time) by reading the DATA port (Offset 4).

d) Write 1 into the Read Complete flag (bit 4). This will allow the next packet to be received.

Transmit Module

The transmit module reads data from a FIFO and transmits it to the PHY. The module will patch in the MAC address that is programmed into the MAC address, it will also patch in the CRC at the end of the packet. The Transmit unit DOES NOT pad packets that re smaller then the minimum size.

The transmit module is design to handle collisions and will follow the standard protocol if a collision is detected. To transmit a packet the software should follow the step

a) Should wait till the transmit module is free (Transmit Busy Flag bit 2 is zero).

b) Write a 1 in the Reset Write Pointer Flag in the Control Register (bit 0).

c) Write the data one byte at a time into the DATA register. The transmit module will auto increment the pointer for each write.

d) Once all the data is written into the FIFO , write a 1 into the Transmit start bit (2) of the control register.

Software support

A basic set of routines to access the EasyMAC registers, to send / receive packets. All the routines assume that a #define symbol EEMAC_BASE is provided in sys_config.h.

`unsigned int eth_get_status()` – returns the control/status register of the EasyMAC.

`void eth_set_status(unsigned int bits)` – Will set the bits specified in the parameter in the control/status register.

`void eth_clr_status(unsigned in bits)` – Will clears the bits specified in the parameter.

`int eth_get_len()` – returns the length field of the control/status register.

`void eth_get_packet(char *buff, int len)` – will copy **len** bytes from the data register (Receive FIFO) into the buffer specified. Will also set the Read complete bit after the data is copied into the receive buffer.

`void eth_send_packet(char *buff, int len)` – will wait for the transmit busy bit to become 0 then copy **len** characters from buff to the transmit FIFO. After the copy is complete it will start transmission by writing 1 to the start transmit bit in the status/control register.

`void eth_set_macaddr(char *maccaddr, int len)` – will set the MAC address to the specified by the macaddr, **len <= 6.**

`void eth_get_macaddr(char *macaddr, int len) –` will read the macc address from the Mac module and place in the buffer specified, len should be <= 6.